UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports Techniques

N° 120

Programme 8
Communication Homme-Machine

DOCUMENT DESCRIPTION LANGUAGE INTERPRESS

Nenad MAROVAC

Juillet 1990

RT-0120

IRISA INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX
FRANCE
Téléphone : 99.36.20.00
Télex : UNIRISA 950 473F
Télécopie : 99.38.38.32

Publication Interne n° 540 - Juin 1990 - 26 Pages

Document Description Language INTERPRESS
INTERPRESS, un langage de description de document

Nenad Marovac
IRISA et San Diego State University

Résumé

Un *langage de description de page* est un langage de programmation permettant de décrire la structure et l'apparence d'une page de document de façon indépendante de toute imprimante. Un *langage de description de documents* spécifie non seulement structure et apparence d'une page, mais aussi de tout un document, son environnement global, les relations entre pages, la composition d'un document à l'aide de pages d'un autre document, etc. Sous-ensemble d'un tel langage, les *spécifications d'environnement d'impression* permettent de préciser l'impression recto/verso, le choix du papier pour une ou plusieurs pages, etc.

Cette note présente ces langages et est une introduction à Intepress, un langage de description de documents. Enfin, il décrit l'expérience acquise lors de l'implémentation d'Interpress sur deux sites.

Summary

A *Page Description Language* is a programming language used to describe the structure and appearance of a document page, in a device (printer) independent way. A *Document Description Language* is a language which is capable not only to specify structure and appearance of a page, but also the structure of a document, the global document composing environment, relationships between pages, combining pages of one document within another document, etc. The *printing environment specification* subset of a language is used to specify printing options like one sided and two sided printing, paper-shift during the printing of a page, tray used to feed in paper to print the entire document or on a page basis, etc.

There are three objectives in presenting this paper. First, to present the concepts of page and document description, as well as printing environment specification languages. Second, to introduce Interpress, a document description language incorporating a printing environment specification subset from Xerox, and third, to describe experience gained in two implementations related to Interpress.

Key words : electronic publishing, page description languages, printers and typeseters, raster imaging processors, and Interpress.

Page Description Language INTERPRESS

Nenad Marovac

IRISA - Campus de Beaulieu[1]
F-35042 Rennes Cedex, France
nenad@irisa.irisa.fr

4 April 1990

Summary

A Page Description Language is a programming language used to describe the structure and appearance of a document page in a device (printer) independent way. A Document Description Language is a language which is capable not only of specifying the structure and appearance of a page, but also the structure of a document, the global document composing environment, relationships between pages, the combination of pages of one document within another document, etc. The printing environment specification subset of a language is used to specify printing options, such as one sided and two sided printing, paper-shift during the printing of a page, and the tray to be used to feed in paper to print the entire document or on a page basis.

There are a number of objectives in presenting this paper. First, the paper introduces the concepts of page and document description, as well as printing environment specification languages. It then presents the structure and functionality of Interpress, a document description language incorporating a printing environment specification subset from Xerox. It illustrates the use of Interpress through an example in the Appendix to the paper. It also gives a brief history of page description languages, and it compares Interpress to Postscript. Finally, it describes the experience gained in two implementations related to Interpress[2].

Key words : electronic publishing, page description languages, printers and typesetters, raster imaging processors, and Interpress.

---

[1] At the time of preparation of this document the author was visiting researcher with IRISA - His regular associations are: San Diego State University (nenad@sdsu.edu) and Xerox (nenad.sd@xerox.com or nenad:sd:xerox), San Diego, USA.

1. Introduction

A *page description language* is a programming language used to describe the structure and appearance of a document page in a device (printer) independent way. Using such a language we could program (encode) an entire page in a document and produce a form which we will refer to as a *page master.* Interpreting this master would produce a page in the appearance we envisaged. An encoded format of the entire document in a page description language is referred to as a *document master.* A page description language is used by an output generation program to encode a document, whether a book, or a computer listing, or a memo, into an output (printer or typesetter) format, to be interpreted by an output device (printer or typesetter or document display device) to produce a visual presentation of the document.

A page description language is made device independent for two reasons. First, we follow the idea that the language should be document oriented rather than device oriented. In other words we formulate the language with the main objective being optimal description of the structure of document pages in terms of primitives, like paragraphs, text lines, rules (graphical lines), images, and similar. Further we incorporate in the language features specifying how these components will be rendered on paper, such as color and thickness of rules, typefaces to be used for different portions of text, and similar. In this we do not tailor the language to make it most suitable and optimal to any particular hardware printing device. However, it is good design to make such language inherently optimal for hardware implementation. This means that the language should tend to incorporate features which can be implemented in imaging (printing or display) hardware in an efficient manner. This will be further dealt with in Section 6.1 of this paper.

Second, the computer community finally realized the need for standardization. This is apparent if one follows efforts in standardization of programming languages, interfaces to graphical software and graphical hardware devices, etc. In fact a page description language should be either a formal or industry standard for a printer interface, i.e. it should be a language which is directly interpreted by printing devices, regardless of manufacturer or model. The importance of this idea is readily apparent to an observer of the electronic publishing market. At the present we can identify three languages which fall into the category of page description languages. They are Interpress from Xerox, Postscript from Adobe Systems, and DDL from Imagen with the first two showing stronger presence on the market. Even after being in a public domain for a very short time, there is a large number of printing hardware and publishing software manufacturers who announced that they will implement either printers, typesetters, or software systems which will interpret or generate one or both of the languages.

A *document description language* is a language which not only specifies structure and appearance of a page, but also the structure of a document, the global document composing environment. Furthermore, it supports combining pages of one document within another document and combining two documents together. The *printing environment specification language subset* is used to specify printing options like one sided and two sided printing, which printer paper feed tray to use to feed paper to print the entire document and/or individual pages, and similar. The printing environment specification, if present, is attached in front of a document master. This specification we will refer to as *environment instructions* or *printing instructions.* Of the three languages mentioned we would classify Postscript and DDL as page description languages, and Interpress as a document description and printing environment specifiction language[1].

This paper is structured into seven sections. The next section, Section Two, includes a functional overview of page description languages, and a more detailed discussion of the functional differences between the three types of languages. It also includes an overview of the Interpress language. Section Three presents the overall structure and functionality of Electronic Printing Systems in general and the Xerox XPS-700 system in particular. Sections Four and Five describe Interpress generator

and Interpress preview sub-systems for the XPS-700 system, respectively. In Section Six we discuss some observations made in generation of Interpress masters, and finally, in Section Seven we briefly make some comparisons between Interpress and Postscript. The Appendix contains excerpts from an Interpress master as an example.

2. Page, document and printing environment description languages

As stated previously, a page description language is a programming language powerful enough to describe the structure of a printed page of any complexity and context, in a device independent manner. Printed pages, in general, are composed from three basic publication primitives. The primitives are text, geometric figures, and sections of photographs. In other words a page comprises one or more columns of text, geometric figures, and sections of photographs in varied gray levels and colors. Therefore, a PDL should incorporate a number of data types and a number of operator,s as well as features to combine data types and operators into composite data types and operators. Basic and composite data types and operators in a page description language must allow for description of a page containing any composition of the basic graphics primitives needed to create a page of a desired complexity. Masters, (programs in encoded in PDL languages) being descriptions of pages and documents, are typically generated and interpreted by machines. They are generated by (composition) programs running on computers for document creation and formating, and interpreted (executed) by programs running in printers, typesetters and display devices[2].

A master is generated automatically by a program as a user interactivelly places text, figures and visual representation of raster scan images on a WYSIWYG screen, or composes a page using a high level markup language. In the second case the user may also preview the page on a screen. In either case the system can produce a document master, when the user indicates acceptance of the context and appearance of the page. In such an environment it seems there is very little need for the user to program a page in a page description language directly, or to proof the master. The functionality of a page description language typically provides for:

- setting a position anywhere within a page
- placing text, with or without justification, within a desired measure
- creation, selection and modification of fonts
- creation of any graphics 2D figures from basic graphics primitives, like line segments, circular arcs, parts of 2nd degree curves, B-splines, closed filled outlines, etc.
- applying any combination of scaling, rotation and translation to graphics figures
- creation, scaling, and rotation of raster scan images

---

[1] To simplify the discussion in the remainder of this paper, except in the next section, we will refer to all three types of languages as page description languages.

[2] Postscript is designed to allow for interactive master generation by users, and ready readability and understandability of document masters in their encoded form by humans. It is believed that such property is more of a disadvantage rather than advantage, because masters in such form are inherently less efficient to interpret by machines.

2.1. Document Descriptiption Languages

A *document description language* should provide all functionality as described above for page description languages as well as additional functionality enabling:

- specification of the structure of an entire document in terms of encoded formats of two or more pages.
- combining pages from two or more document masters into one document
- merging together a part of a page from one master in one or more instances with a page from another master. This is useful when a part of a page is an illustration generated at an illustrator's workstation that has to be pasted at different positions on a currently composed page.
- concatenation of two or more complete masters to be printed together as a unit.

2.2. Printing Environment Specification Languages or Publishing Environment Integration Languages

Such a language should possess all functionality of document specification languages with some added system functionality. Since at this point only one such language exists, e.g. Interpress from Xerox, Interpress will be used to explain additional features of such languages[1]. Interpress sopports:

- Printing instructions
  - printing instructions for the entire document
  - printing instructions on a page level
- Complete font encoding
  - Typefaces - Font metrics
  - Font distribution mechanism
  - User font definition
- Encoding of bit-map images
- File insert mechanism
- In-line file insert
- Local file insert
- Remote file insert

Printing instructions. This feature allows a document creator to add to the document master additional information which will either be printed with the document, like document creator name, date of creation, etc., or will specify the printing environment, like color of the paper to be printed on, one sided or two sided printing, stapling of documents, and similar. These printing environment instructions can be specified either on the document level for the entire document as a whole, or on a page level for each page prior to the section of the master for the page. Interpress allows for printing environment instructions on the document level to be overwritten when the document is actually sent to be printed.

Font encoding. Interpress is used to encode complete information about a font, including font metrics, font bit maps or contour structures for the font. For each font there is one Interpress master. This format of fonts is used for font distribution[2] to customers, either for their printing machines or for their document composition systems. Fonts are also to be stored in this format in font servers in office environments, and when a printer needs a font to render a document it will fetch it from the system font server. Furthermore, Interpress provides the means for a user to define fonts within document masters.

Encoding raster scan images. Scan images produced by scanners, or any other devices in the office environment are encoded into Interpress masters[3]. This allows for general and generic merging mechanisms of documents and their parts into documents. Images produced by Xerox scanners and the EPIC system are encoded as Interpress masters.

File insert mechanisms. Documents and images to be merged into another document can be merged in-line into a document during its creation and encoding. Alternatively, the documents to be merged may already reside on the printer, or be stored in a network server of the office environment. Interpress provides a mechanism for these documents to be recalled either from the printer disk storage, or from the office network server, during the printing of the document containing these documents or images.

Below are listed Interpress properties as perceived from a user's point of view:

- Functional richness
- Multi-layer organization
- Explicit and formal document structuring
- Page independent structure
- Device independence
- Priority important
- Compact encoding
- Performance
- Total environment
- Printing instructions
- Extensibility

The multi-layer organization or subsetting of Interpress has raised some controversy. Dividing a language into a hierarcy of functional subsets, forming a base for implementation alternatives, does present some problems. A master may incorporate some constructs which are not supported by a local printer. Images described by these constructs may not be rendered correctly at that printer. However, the printer should at least issue a warning message, and print the rest of the document correctly. This is not as difficult to ensure as it may seem. However, there is the other side of the coin also. The layer approach allows both for more efficient and the cheaper implementations of subsets. A printer designed and implementing text and graphics figure primitives at black and white levels only, will be considerably cheaper and can be made faster than a printer implementing full Interpress supporting color. Experience with similar standards, such as GKS[4] seem to enforce this fact.

Interpress uses binary encoding. The encoding is compact and designed for efficient decomposition of document masters. This is an important factor in designing and implementing printers with a high throughput and large volume publishing, like the Xerox 9700/8700 printer family.

Some additional observations and suggestions for the generation of efficient Interpress masters are included in Section Six of this paper.

2.3. Language Interpress - A Brief Overview

It is not our intention to present here a detailed description of the language Interpress. This can be found in the reference[1]. The Appendix also contains excerpts from a sample of an Interpress program. Here we wish only to give a very brief overview of an Interpress document master and the functionality of the language.

Page Description Language INTERPRESS

A description of a document encoded in Interpress is called an Interpress master for that document. An Interpress master has a very formal structure and it contains four components:

- Header
- Printing Instruction section
- Preamble section
- Page Master sections

The *header* simply states that this is an Interpress program and specifies the Interpress version.

All master sections are enclosed within braces. The *printing instruction* section is optional. It contains specification for printing of the document, e.g. paper size, paper color, one side or two sided printing, finishing like stapling, etc. The *preamble* section specifies part of document environment which is common throughout the document, e.g. fonts and similar.

A *page master section* describes in Interpress encoding the content and appearance of a page. There is one page master for each page. In front of a page master might be a *page printing instructions* section (formally referred to as content instructions) specifying printer processing for that page, e.g. change paper feed tray with different paper color, shift paper vertically, etc. Some page printing instructions affect just that page, and some affect that page and all following pages until another page printing instructions section changes the specification.

An Interpress master has the following format:

Header
{PrintingInstructions}
BEGIN
{Preamble}
CONTENTINSTRUCTIONS {PagePrintingInstructions}{PageMaster} or {PageMaster}
CONTENTINSTRUCTIONS {PagePrintingInstructions}{PageMaster} or {Page Master}
...
CONTENTINSTRUCTIONS {PagePrintingInstructions}{PageMaster} or {PageMaster}
END

As mentioned previously Interpress uses binary encoding, and an encoding in an Interpress master resembles encoding in a machine code program with variable length machine instructions. An operator occupies one or two bytes. The most frequently used operators take one byte and the others two bytes. An operand may take two bytes (for a short number) to any number of bytes, as required. See[1] pages 15 and 16.

Operands may be:

- short number
- short sequence
- long sequence

A sequnce has a type which may be:

- String
- Integer
- InsertMaster
- Rational
- Identifier
- Comment

- Continued
- LargeVector
- PackedPixelVector
- CompressedPixelVector
- InsertFile
- AdaptivePixelVector

Interpress supports six types of objects. They are:

- Number
- Identifier
- Mark
- Vector
- Body
- Operator

where a *Mark* is a "synchronization" flag inserted on the top of Interpres system stack, a *Vector* is a generalized heterogeneous array (record), and a *Body* is similar to a procedure body in a programming language. Interpress has three inherent data structures. They are:

- Stack
- Frame
- ImagerVariables

The *stack* is the normal LIFO structure found in machines processing block oriented languages. The *Frame* is a structure of 50 elements. Any element of the Frame structure can be of any Interpress type at any time. The Frame is the only structure which retains values from preamble to page masters. The ImagerVariables reflect imager state, like current color, current position, etc.

Interpress language uses postfix notation, similar to Forth.

2.4. History of Page Description Languages

Interpress and Postscript have a common history tree. The reason for this is very simple: the people who worked on Interpress while employed by Xerox, left Xerox to found Adobe and develop Postscript.

Bob Sproul while at Xerox led an effort in developing a printing format called *Press*. It is used by Xerox printers in a network environment on experimental 3Mb/s Ethernet.

When John Warnock came to Xerox PARC he joined printing format development effort, and together with Marty Newman developed a language called JaM, standing for John and Marty. JAM had strong similarity to the language Forth which John Warnock participated in developing before coming to Xerox.

Finally, a group under Chuck Geschke including Bob Sproul, Butler Lampson, John Warnock, Brian Reid and Bob Ayers developed what then became known as Research Interpress[7].

In 1982 became a Xerox Printing Standard[1].

It was first incorporated into a commercial Electronic Publishing System product in 1984 by the author[8].

Chuck Geschke and John Warnock left Xerox, started Adobe and developed the language Postscript and a commercially available Postscript decomposer which was installed on a number of platforms.

3. Electronic Publishing Systems

An Electronic Publishing System (EPS) is an integrated publishing system for producing documents on demand, which incorporates computers, workstations, electronic printers, and digitizing scanners to input graphics. The use of such a system in producing documents encompasses four activities:

3.1. Text preparation

In this activity a writer prepares the text of the document, which is then typed into a word processor or a computer. He also identifies the drawings or artwork to be included in the document.

At this stage style specifications for the document are prepared. Style specifications define the layout of the document, i.e. they are composed of instructions to the composition and printing sections of the EPS instructing the latter how to process and print the document in accordance with the user's needs. The style specification includes indications for the kind of typefaces to be used for different sections of the document, settings for margins and column widths, the locations of chapter and section headings and the inclusion of previously generated and stored information such as forms, graphics and digital signatures. The style specifications are either prepared explicitly for the document or a previously generated one is identified and incorporated within the document.

3.2. Graphics input and preparation

There are two major sources of graphics images in electronic publishing: computer-generated images and images generated from hand-drawn art-work or photographs.

Typically, in the past a user allocated spaces within the text to insert drawings and camera-ready artwork. More recently the user would generate digitized forms of artwork and photographs by using digitizing scanners like the Xerox 150 Graphics Input Station (GIS) to scan artwork, or computer programs to produce raster scan formats of drawings specified in mathematical forms. In either case the digital images are sent to the computing element of the EPS where they can be stored and directly used by the composition section of the EPS, like Xerox XICS[5], for merging and printing within the document.

3.3. Document composition

In this activity the document is formatted into elements that compose a page as we know it. During this formatting process the composing task deals with running heads; hyphenation and justification; change bars; selection and placement of tables, graphics, and captions, etc. This processing is done according to style specifications.

In the Xerox Publishing System XPS-700 [6], the composition process of a document is actually divided into two sub-tasks. The first sub-task is the composer or composition task proper, called COMPUSET, which processes the document and produces a so called intermediate document format

which is in a machine independent format. The second sub-task is composed of one or more generators. A generator produces output which is very machine dependent, and in fact is a translator from the intermediate format of a document into an output format designated for a particular printer or typesetter. Therefore, it resembles a code generation section of a compiler. In a generator the final formatting of each page is made by determining such things as orientation of the final printed sheet, merging of graphics images and boilerplates with the text, etc. Xerox supports as many generators as there are different makes and models of printers and typesetters to be supported with the EPS.

One of these generators, called XIPINT, is for Interpress. It is also referred to as the Interpress Generator for XPS-700. It translates the output from COMPUSET for a document into the Interpress master for that document, the Interpress master being the output format for that document understood by Xerox printers. The Interpress master is all that is needed to print the document on any of those printers. Therefore, for our purpose COMPUSET and XIPINT together form a full composition section for the Xerox XPS-700 system.

3.4. Printing

The print section of an EPS may be comprised of a print machine like the Xerox 9700/8700/4050. It will print a document including merging and collating of text and graphics, instantly and automatically at rates of up to 120 pages per minute.

The structure of an EPS and relationships between four activities in producing a document are shown in Figure 1. More recent EPS, like the Xerox XPS-700, also include a page design studio activity. Through this activity using a WYSIWYG display, the user can design the structure and the appearance of a page. This activity makes use of two tasks: the composition tasks and the screen preview task. The preview task generates display dependent format for the page to be displayed. The structure and functionality of the preview task is very similar to the structure and functionality for previously discussed generators.

The structure of XPS-700, as shown in Figure 2, has three further additions. They are: the library task, the receiving task, and the Interpress preview task. The document library DCLIBs store documents in Interpress format. These documents are generated using the XPS-700, a document preparation workstation (like VIEWPOINT/STAR or similar), scanning devices (like Xerox 150 GIS), or artwork stations (like Xerox Publishing Illustrators Workstation). These documents can be later printed as individual documents, or merged with other documents in preparation. When using an Interactive design studio, we might preview a document which contains one or more other merged documents and images in Interpress format, therefore the screen preview activity contains an Interpress preview subsection.

At this point we see clearly another use of Interpress language. It plays the role of an universal language functionally linking together all devices on the network. It allows for information exchange between different types of devices on a network in an office environment in an uniform format. It allows scanned images, which must be clearly encoded in a format different from text, to be merged with text at document composition. It also allows documents, generated from any other source on the network, to be combined together, etc. Fonts and their metrics are distributed to printers and any other devices needing them, encapsulated in Interpress format. This feature of Interpress makes it unique when compared with other page definition languages. We referred to Interpress previously as a document description language incorporating a printing environment specification. At this point it would be more precise to refer to it as a language for the *Office and Publishing Environment or Publishing System Integration Language.*

4. Interpress Generator for XPS-700 (XIPINT)

As stated earlier XIPINT is the code generation section for the composition activity in the Xerox XPS-700 publishing system. Composition calculations and decisions are made in the composition front end called COMPUSET. The results from COMPUSET are a set of page specification directives in the form of an intermediate language.

XIPINT then sets the environment for each page and the entire document. It determines necessary transformations for each page as a whole, and for each individual item. It fetches documents and images to be merged (pasted) in the preparation of a document, and sets a document by generating Interpress code for a printer or a typesetter.

It provides functionality to:

- Generate the Interpress representation of a document
- Incorporate printing instructions on request
- Support head-to-head and head-to-toe printing
- Support all four printing orientations
- Support saving of documents in Interpress format (output from XIPINT or any other Interpress generator) into Document Libraries
- Support merging of scanned images within documents in all orientations
- Support merging documents (in Interpress formats) within a document currently being processed. Merged documents can be from any Interpress source (e.g. XPS-700, VIEWPOINT/STAR, 860, etc)
- Support simplex and duplex (one sided and two sided) printing modes
- Support paper feed from different trays for large printers
- Support printing documents on all Xerox network Interpress printers, e.g. 9700, 8700, 4050, 8044, etc
- Support multiple imposition printing (signatures).

In designing XIPINT one of the objectives was to use the smallest complete subset of Interpress which would incorporate the cheapest, i.e. the most time efficient, Interpress operators. At the time of initial design for XIPINT, Interpress was in its infancy and various Interpress printers available at that time supported different subsets of Interpress operators with varying efficiency. Since documents composed in XPS-700 had to be printed on every Interpress printer, research was conducted to determine a common set of operators supported at that time. Research also determined a likely subset that would be supported by every possible Interpress printer in a near future to offer guaranted printing within the XPS-700. Furthermore, printing speed and throughput of different printers differed considerably. In order to guarantee a certain printing performance by a cross section of available Interpress printers, an additional study was conducted to identify which operators were inherently more efficient and tended to be optimized in most implementations of Interpress printers. The result of these two studies determined the subset of Interpress operators generated by XIPINT.

XIPINT architecture is illustrated in Figure 3. It has three levels of implementation. The first level, the composition and functionality level, is application dependent. It reflects constructs used in our composition activity. The second level, the Interpress functionality level, reflects the philosophy of Interpress. The third level, the basic Interpress operator level, implements each individual operator separately. This division resulted from two objectives: easy maintenance of the system, and reusability of individual modules from different levels in other projects. XIPINT was implemented in FORTRAN IV. The reason behind this was to make it fast, and more important to make it transportable. Originally, XIPINT was part of the XICS system. XICS runs on a very large number of

different machines under various operating systems. At that time (1984) only FORTRAN provided for a real portability across main frames, mini computers and micro computers with different word sizes and different operating systems with different filing systems.

5. Interpress preview for XPS-700 (DIPINT)

This task is a decomposer and a software RIP (Raster Imaging Processor) for Interpress. An Interpress RIP accepts a document Interpress master. It decomposes the master and produces code to render the document on a typeseter or printer. DIPINT performs the same function with the difference that it renders the document on a screen in order to preview a document which is already in Interpress format. Such a preview is done for two reasons. One reason is to examine the document before sending it to a printer. The second reason, more important in XPS-700, is to preview a document or image when merging it with the document being designed and composed interactively at a WYSIWYG terminal. Typically, these documents and images to be merged come from different sources, all of which have only Interpress format in common.

DIPINT organization is shown in Figure 4. It shows three levels. They are: the Interpress machine level, the Interpress operators level, and the graphics imaging level. The first level contains modules to parse the language, including the decomposition section in a form of a finite state machine. It processes a master and maintains Interpress structures: stack, Interpress vectors, frame and imager data structures. The second level includes modules implementing each individual Interpress operator. The third level contains graphics modules to actually do all the imaging necessary to render a document. This architecture again resulted from the same two objectives as discussed in XIPINT, i.e. maintainability and reusability. Ideally, to use DIPINT to implement an Interpress machine of another type such as a printer, only the third level routines need to be redesigned and recoded. Also, to be compatible with graphics hardware of tomorrow, the functionality and calling sequences of graphics modules were modeled as close as was feasible to CGI and GKS.

6. Experience with Interpress in EPS environment

Time spent in building and testing XIPINT was definitively exciting. It required involvement in many related activities. At that time high quality publishing fonts, like Mergenthaler's Optima, Helvetica and similar did not exist under Interpress. Therefore, one of the first tasks was to introduce these fonts into Interpress on 9700 series machines. However, these fonts did not exist on other Xerox Interpress machines like the 8044. The 8044 does an automatic font substitution, but the substituted font typically has different metrics. This could potentially degrade the quality of a printed page. To solve this problem use of the CORRECT Interpress operator, which is defined exactly for this purpose, was very valuable. However, the CORRECT operator is usually implemented on printers as a two pass operator, which makes it very expensive to use. In order to obtain the best compromise, an option was introduced into XIPINT allowing the user to decide whether to use CORRECT for justification (particularly in a font substitution environment) or to rely on the precision of the composer assuming no font substitution will be needed. In the second case the document will be printed much faster, an obvious advantage in large volume printing.

6.1. Optimization of Interpress masters

It was thought best to conclude this section with some observations about Interpress master generators and optimization of Interpress masters. These observations result from involvement in both the XIPINT and the DIPINT projects.

In the generation of Interpress masters two optimization criteria can be followed:

- Size (encoding efficiency)
- Printing (decomposition and imaging) speed

The first criteria is generally important for two reasons. First, to reduce the size when we wish to keep the entire master in main memory while decomposing it to reduce disc access time. This is relevant for smaller printers which are dedicated mainly for printing of smaller documents, and may also be relevant for large printers for fast printing of large documents. Second, to reduce the communication time when sending the documents in Interpress format to a printer or to another location. However, the constantly decreasing price of memory, combined with increasing addressing space of microprocessors and the larger bandwidth of local networks, makes this criteria less important than the second one, i.e. printing speed.

Printing speed is very much dependent on the content of Interpress masters. It is possible to formulate a number of rules, which if followed would result in more efficient Interpress masters. However, the most important optimization strategy for generating Interpress masters is based on basic common sense. A document is processed by a composer-printer pair. The real question is where does the composer end and where does the printer begin? Let's look at two very simple examples.

First, suppose that we need to plot a complex curve on a page of a document. Do we approximate this curve into line segments within the composer and request a printer to render the line segments? or, do we construct a sequence of second degree curves and B-splines approximating the desired curve, and make the printer approximate these second degree curves and B-splines curves by line segments in order to render the desired curve.

Second, suppose again that we wish to produce a justified paragraph on the page. Do we do all required calculations within the composer and relatively position all words so that if the printer simply positions and paints the words, the paragraph will automatically be rendered justified? Or do we instruct the printer to put certain words into a certain measure to produce justified line of text which will again result in a justified paragraph?
In both cases we come to the conclusion: let the Interpress generator do as much work as possible to reduce the printer's work. This is correct for two reasons. First, a composer running on a main frame or a mini computer will have at its disposal a much better floating point arsenal than a printer in general. Sophisticated floating point hardware will make small and cheap printers too expensive. Second, a document may be composed once and printed many times in many copies on request.

This argument is similar to the implementation of compilers. An Interpress generator is no different really from a code generator in a compiler, and a printer from computer hardware for execution of programs. Do we wish more work to be done at compilation of a program, and less at its execution, or vice versa?

These arguments do not lead to the conclusion that we should build less sophisticated printers. It just points to a more efficient division of labor in a composer-printer pair in an EPS. This is particularly relevant when we have a number of different printers with different degrees of processing speed connected to an EPS.

7. Interpress and Postscript

Since Interpress and Postscript are two major Page Description languages in use today it seems natural to devote a few words to compare them. We will start first with listing their common features.

7.1. Similarities

Interpress and Postscript have following similarities:

- Both are PDLs
- Both use Forth like postfix notations
- Both are device independent
- Both contain two parts: general purpose programming part and imaging part
- Both use stack oriented block processing structure
- Both use byte oriented stream to represent document to output device
- Both use generalized array processing capabilities
- Both use universal coordinate system and identical forms of transformation matrix
- They have very similar imaging models

Now let's look at some features in which Interpress has the advantage over Postscript.

7.2. Differences - Interpress strength

Interpress has advantage over Postscript in the following:

- Total System (Office) Environment
  - SequenceInsertFile
  - Printing Instructions
- Page Independence
- Well Defined Structure
- Printer Instructions
- Priority Important
- Compact Encoding
- Compression Techniques for Bit-Map Images

Three of the above mentioned features merit further eleboration.

Page Independence

In Interpress pages are syntactically clearly separated. A page startswith a "{" and ends with a "}". Furthermore, each page carries its own environment. The only environment (contents of the Frame structure) a page inherits is the portion set in the preamble of a document. At the beginning of each page the environment is reset to the environment at the end of the preamble. The syntactical and environment separation of pages makes it very simple to separate pages from a document when saving the document in a document library data base, in its Interpress format. Then any page from a document can be merged at request into another document being encoded into the Interpress format.

Compact Encoding and Compression Techniques for Bit-Map Images

This feature is more important than it may seem at first. A Postscript document of about 70 pages can easily occupy about 5Mb of disk space. This makes it tedious to send documents in Postscript form via a network. This is particularly true if the document contains large bit map images. In this case, when protocol does not allow for 8-bit binary data transfer - which is true in implementation of Postscript printers and drivers - each byte of an image has to be split into two nibbles and transmitted as two bytes. The amount of data to be transmitted can grow very fast. Interpress compact binary encoding can reduce the amount of data by a factor of 3 to 5. With compression encoding of bit map images, the data reduction factor can increase dramatically.

Finally, let's look at features in which Postscript has advantages.

7.3. Differences - Postscript strength

Postscript has advantages over Interpress in the following:

- File Handling
- General Purpose Programming Capability
- More General Procedure Calling
- Graphics Imagiing Capabilities

Conclusion

Concepts of page definition languages were presented. A closer look at Interpress in an Electronic Publishing Environment was made. Two projects related to Interpress were discussed, and finally some observations from experience in implementing an Interpress generator and a software Interpress RIP were given.

Acknowledgements

I would like to thank Jacques Andre for inviting me to visit and work at INRIA-IRISA at Rennes, and INRIA and Ministry of Research and Technology of France for their support during my stay. I also wish to thank Elizabeth McClure for proof reading the paper.

The bibliography

[1] INTERPRESS - Electronic Printing Standard, Xerox Corporation.

[2] Font Interchange Standard, Xerox Corporation.

[3] Raster Encoding Standard, Xerox Corporation.

[4] ISO/DIS 7942 - 1982 Information Processing: Graphical Kernel System (GKS).

[5] The Xerox Integrated Composition System (XICS). Reference Manual, Xerox Corporation.

[6] Xerox Publishing System. System Description Manual, Xerox Corporation.

[7] A device independent graphics imaging model for use with raster devices, Computer Graphics, Vol. 16, No. 3, July 1982, pp. 313-320.

[8] Nenad Marovac, Page Description languages: Concepts and Implementations, Workstations and Publication Systems, Ed. R. A. Earnshaw,Springer-Verlag 1987.

Appendix

The following code representa excerpts from an Interpress master. The first portion includes the *header* and the *printing instruction section*. The printing instruction section contains:

- comment paragraph stating that this is a XPS print job
- media subsection containing three media specifications, one for each paper feed tray (assuming this is for a three tray device). Each media specification paragraphs states:
  - paper X and Y size in metres
  - paper color.
  - transparent or opaque printing media
  - paper prefinish, e.g. plain, 3 hole drilled, etc.
- preselection of media stating that the entire document is to be printed using media from feed tray one
- document to be printed two sided
- the document should be corner stappled
- it should be named at print as "JUDY DSAV"
- the document is created by the user called "SYSTEM"

```
Header: "Interpress/Xerox/2.2 "
{
> Identifier: docComment
> String: "XPS700 - PRINT JOB"
> Identifier: media
> Identifier: mediumXSize
> Rational: 2159/10000 (0.215900)
> Identifier: mediumYSize
> Rational: 2794/10000 (0.279400)
> Identifier: color
> Identifier: buff
> Identifier: opacity
1
> Identifier: preFinish
> Identifier: plain
10
makevec
> Identifier: mediumXSize
> Rational: 2159/10000 (0.215900)
> Identifier: mediumYSize
> Rational: 2794/10000 (0.279400)
> Identifier: color
> Identifier: goldenrod
> Identifier: opacity
1
> Identifier: preFinish
> Identifier: plain
10
makevec
> Identifier: mediumXSize
> Rational: 2159/10000 (0.215900)
> Identifier: mediumYSize
```

```
> Rational: 2794/10000 (0.279400)
> Identifier: color
> Identifier: yellow
> Identifier: opacity
1
> Identifier: preFinish
> Identifier: plain
10
makevec
1
3
makeveclu
> Identifier: mediaSelect
1000
1000
1
2
makevec
2
makevec
> Identifier: plex
> Identifier: duplex
> Identifier: finishing
> Identifier: cornerStaple
> Identifier: docName
> String: "JUDY DSAV"
> Identifier: docCreator
> String: "SYSTEM"
14
makevec
}
```

The second portion is a preamble which in this case does not do much. It just sets a Modern font as the current font.

```
BEGIN (block)
{
1
16
iset
> Rational: 1/56693 (0.000018)
scale
1
fset
> Identifier: XEROX
> Identifier: XC1-1-1
> Identifier: MODERN
3
makevec
findfont
200
scale
modifyfont
```

```
0
fset
0
setfont
}
```

What follows bellow are masters for two pages. Both pages are identical. The first paragraph in each page sets the transformations for the page (Interpress works in metres). The paragraph also sets the tolerance for the CORRECT operator (CORRECT operator forces a text string to be rendered within a predefined measure). Finally, the paragraph also sets the stroke ends to be of type 1 (butt) and stroke widths to be 2 units. Note that units are set by the scale operator to be (0.000035 m or 1/10 printer point).

```
{
> Rational: 35278/1000000000 (0.000035)
scale
concatt
5
5
setcorrecttollerance
1
16
iset
2
15
iset
```

The next paragraph renders a vertical line (stroke).

```
1061
7000
moveto
6560
linetoy
```

The next paragraph will set current font to be Classic 10 printer point, and it also sets (show) a character string. It then ends the page with "\}". What follows after is the identical structure for the next page. Finally, we have the end of the document indicated by "END". It is a rather boring document, but hopefully it illustrates the structure of Interpress masters.

```
1060
6880
setxy
> Identifier: XEROXo
> Identifier: XC1-1-1
> Identifier: CLASSIC
3
makevec
```

```
findfont
100
scale
modifyfont
1
fset
1
setfont
> String: "Level"
show
}
{
> Rational: 35278/1000000000 (0.000035)
scale
concatt
5
5
setcorrecttollerance
1
16
iset
2
15
iset
1061
7000
moveto
6560
linetoy
maskstroke
1060
6880
setxy
> Identifier: XEROX
> Identifier: XC1-1-1
> Identifier: CLASSIC
3
makevec
findfont
100
scale
modifyfont
1
fset
1
setfont
> String: "Level"
show
}
END (block)
```

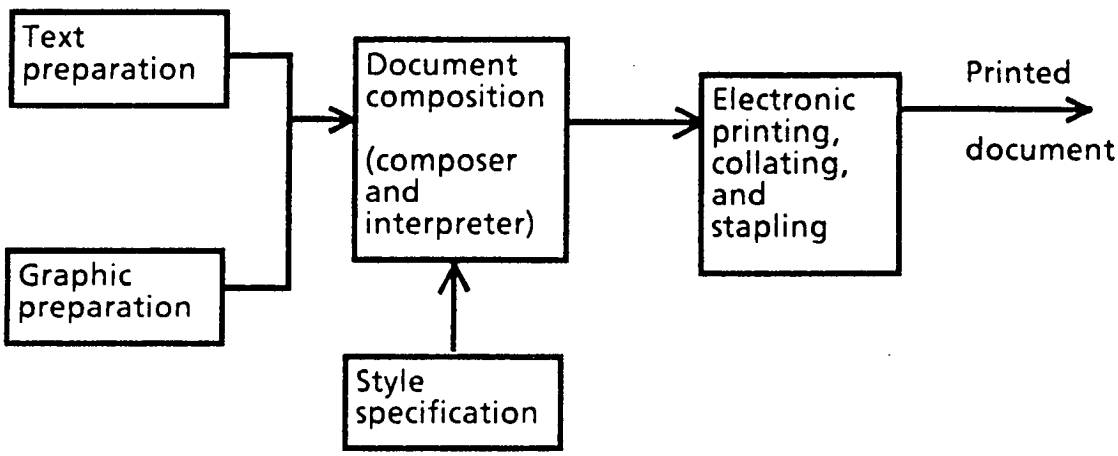

Figure 1. Structure of an Electronic Publishing System

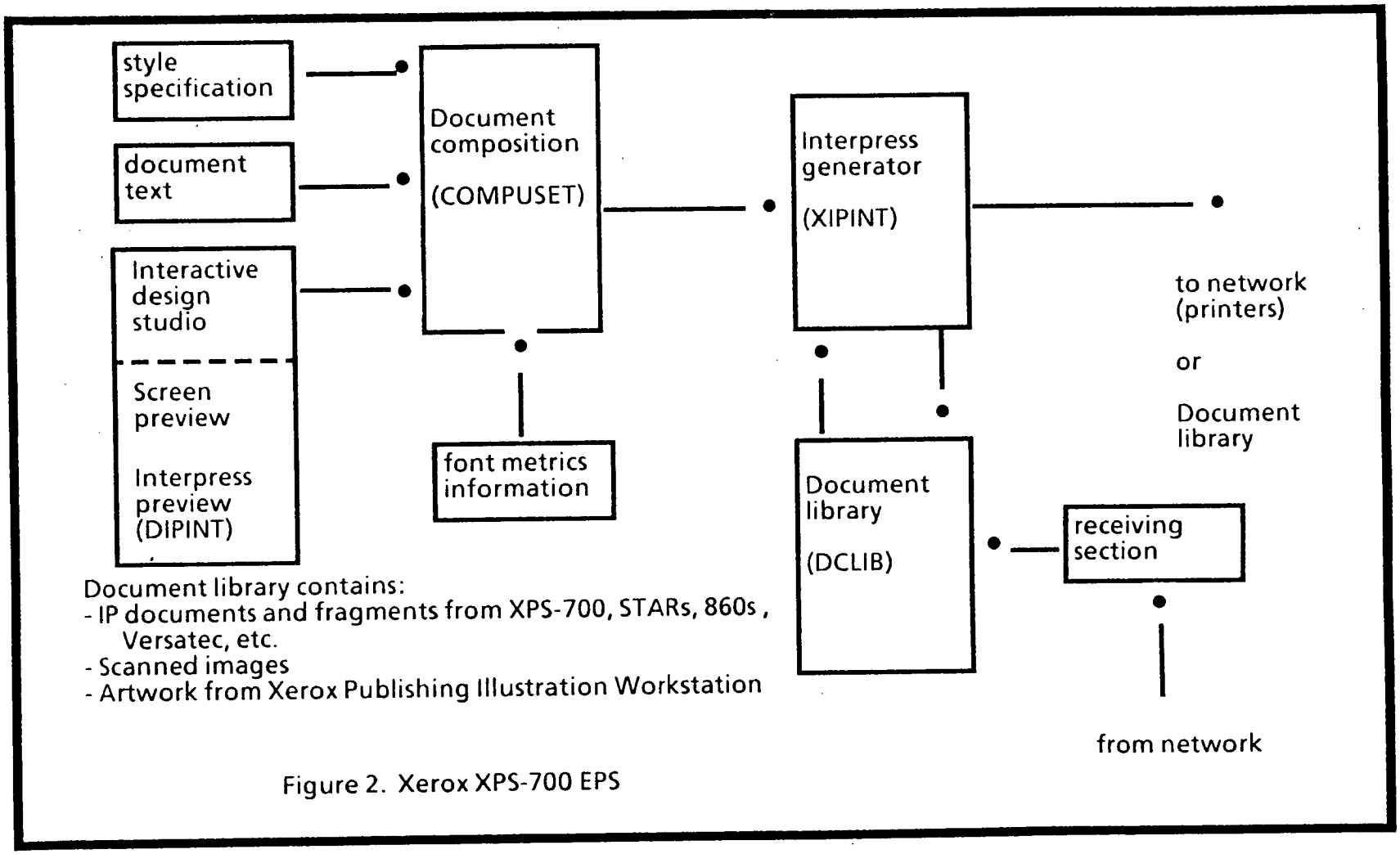

Figure 2. Xerox XPS-700 EPS

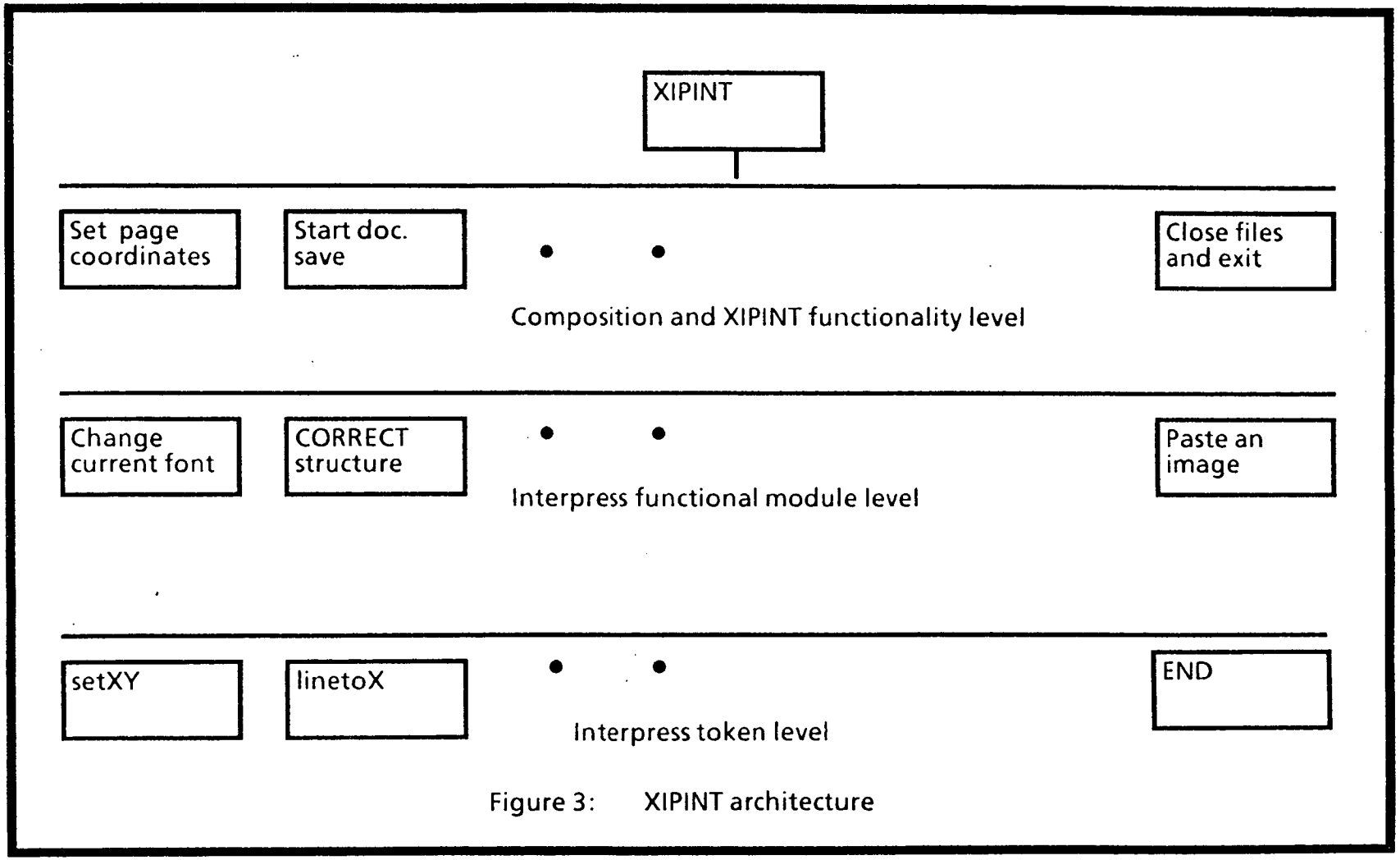

Figure 3: XIPINT architecture

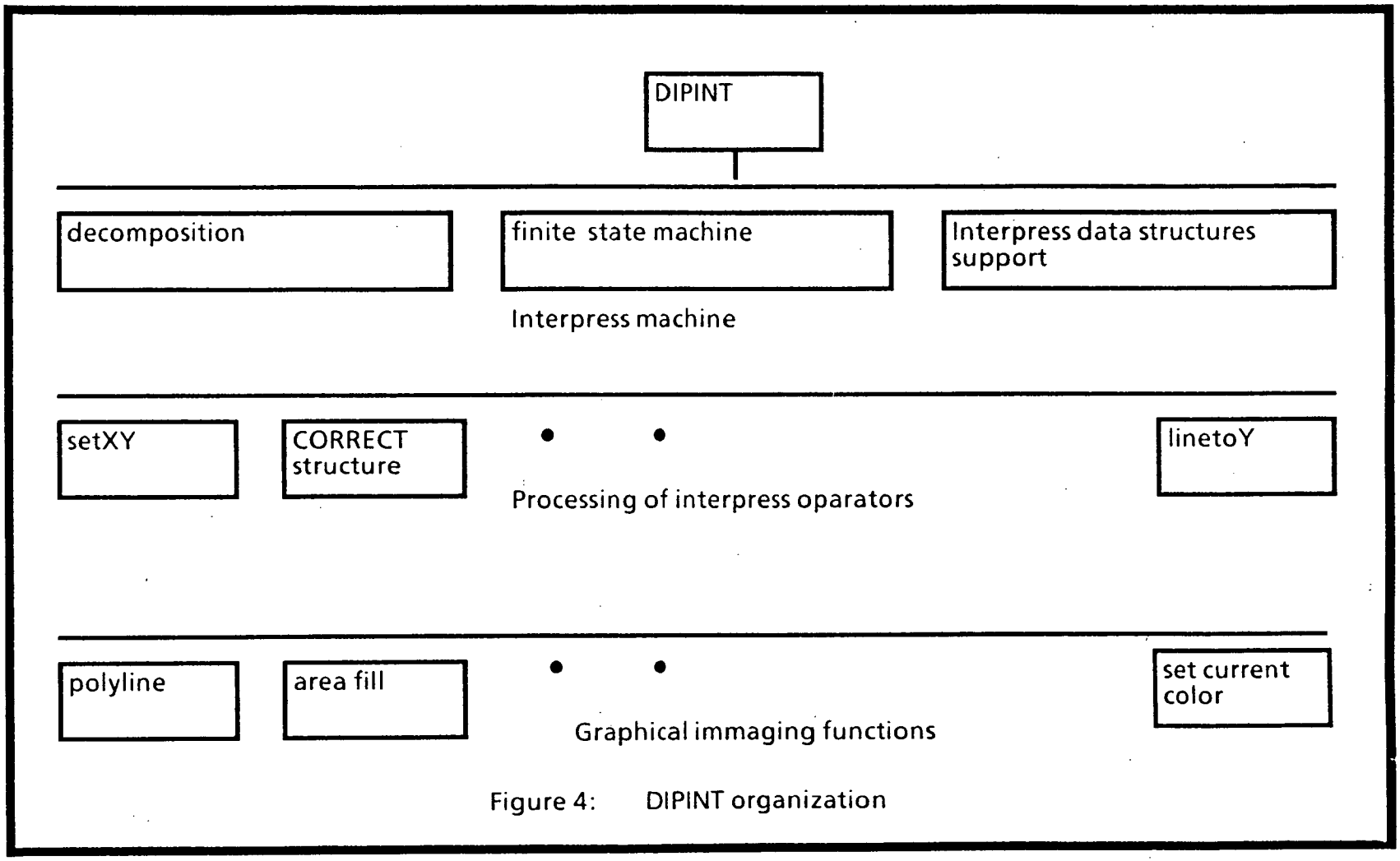

Figure 4: DIPINT organization

LISTE DES DERNIERES PUBLICATIONS INTERNES

PI 530 **SEMI-GRANULES AND SCHIELDING FOR OFF-LINE SCHEDULING**
Bernard LE GOFF, Paul LE GUERNIC, Julian ARAOZ DURAND
Avril 1990, 46 Pages.

PI 531 **DATA-FLOW TO VON NEUMANN : THE SIGNAL APPROACH**
Paul LE GUERNIC, Thierry GAUTIER
Avril 1990, 22 Pages.

PI 532 **OPERATIONAL SEMANTICS OF A DISTRIBUTED OBJECT-ORIENTED LANGUAGE AND ITS Z FORMAL SPECIFICATION**
Marc BENVENISTE
Avril 1990, 100 Pages.

PI 533 **ADAPTATION DE LA METHODE DE DAVIDSON A LA RESOLUTION DE SYSTEMES LINEAIRES : IMPLEMENTATION D'UNE VERSION PAR BLOCS SUR UN MULTIPROCESSEUR**
Miloud SADKANE, Brigitte VITAL
Avril 1990, 34 Pages.

PI 534 **DIFFUSE INTERREFLECTIONS. TECHNIQUES FOR FORM-FACTOR COMPUTATION**
Xavier PUEYO
Mai 1990, 28 Pages.

PI 535 **A NOTE ON GUARDED RECURSION**
Eric BADOUEL, Philippe DARONDEAU
Mai 1990, 10 Pages.

PI 536 **TOWARDS DOCUMENT ENGINEERING**
Vincent QUINT, Marc NANARD, Jacques ANDRE
Mai 1990, 20 Pages.

PI 537 **YALTA : YET ANOTHER LANGUAGE FOR TELEOPERATE APPLICATIONS**
Jean-Christophe PAOLETTI, Lionel MARCE
Juin 1990, 32 Pages.

PI 538 **SYNCHRONOUS DISTRIBUTED ALGORITHMS : APROOF SYSTEM**
Michel ADAM, Jean-Michel HELARY
Juin 1990, 20 Pages.

PI 539 **CONCEPTION DE DESCRIPTEURS GLOBAUX EN ANALYSE DU MOUVEMENT A PARTIR D'UN CHAMP DENSE DE VECTEURS VITESSES APPARENTES**
Henri NICOLAS, Claude LABIT
Juin 1990, 38 Pages.

PI 540 **DOCUMENT DESCRIPTION LANGUAGE INTERPRESS**
Nenad MAROVAC
Juin 1990, 26 Pages.

Imprimé en France
par
. l'Institut National de Recherche en Informatique et en Automatique .

ISSN 0249 - 0803